Shortest Paths with Negative Weights

Slides by Carl Kingsford

Feb. 11, 2013

Based in part on Section 6.8

Shortest Path Problem

Shortest Path with Negative Weights. Given directed graph G with weighted edges $d(u, v)$ that may be positive or negative, find the shortest path from s to t.

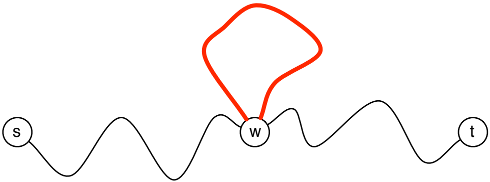
Complication of Negative Weights

Negative cycles: If some cycle has a negative total cost, we can make the $s - t$ path as low cost as we want:

Complication of Negative Weights

Negative cycles: If some cycle has a negative total cost, we can make the $s - t$ path as low cost as we want:

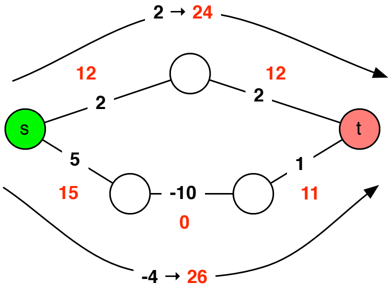
Go from s to some node on the cycle, and then travel around the cycle many times, eventually leaving to go to t.



Assume, therefore, that G has no negative cycles.

Let's just add a big number!

- Adding a large number M to each edge doesn't work!
- The cost of a path P will become $M \times \mathrm{length}(P) + \mathrm{cost}(P)$.
- If M is big, the number of hops (length) will dominate.



Bellman-Ford

Let $dist_s(v)$ be the current estimated distance from s to v.

At the start, $dist_s(s) = 0$ and $dist_s(v) = \infty$ for all other v.

Ford step. Find an edge (u, v) such that

$$dist_s(u) + d(u, v) \leq dist_s(v)$$

and set $dist_s(v) = dist_s(u) + d(u, v)$.

Repeatedly Applying Ford Step

Theorem. *After applying the Ford step until*

$$dist_s(u) + d(u, v) \geq dist_s(v)$$

for all edges, $dist_s(u)$ will equal the shortest-path distance from s to u for all u.

Proof. We show that, for every v:

- There is a path of length $dist_s(v)$ (next two slide)
- No path is shorter (in three slides)

So $dist_s(v)$ must be the length of the shortest path. □

A path of length $dist_s(v)$ exists

Theorem. *After any number i of applications of the Ford step, either $dist_s(v) = \infty$ or there is a $s - v$ path of length $dist_s(v)$.*

Proof. Let v be a vertex such that $dist_s(v) < \infty$. We proceed by induction on i.

Base case: When $i = 0$, only $dist_s(s) = 0 < \infty$ and there is a path of length 0 from s to s.

Induction hypothesis: Assume true for all applications $< i$.

A path of length $dist_s(v)$ exists, II

Proof, continued.

Induction step: Let $dist_s(v)$ be the distance updated during the ith application. It is updated using some edge (u, v) using the rule:

$$dist_s(v) = dist_s(u) + d(u, v)$$

$dist_s(u)$ must be $\leq \infty$ and thus must have been updated by some application of the Ford rule at a step before i.

Therefore, by the induction hypothesis, there is a path P_{su} of length $dist_s(u)$.

Now, on the ith application $P_{su} + (u, v)$ is a path of length $dist_s(u) + d(u, v) = dist_s(v)$ □

No paths are shorter

Theorem. *Let P_{sv} be any path from s to v. When the Ford step can no longer be applied, $length(P_{sv}) \geq dist_s(v)$.*

Proof. By induction on # edges in P_{sv}.

Base case: When $|P_{sv}| = 1$, it consists of a single edge (s, v) and because the Ford step can't be applied $d(s, v) \geq dist_s(v)$.

Induction hypothesis: Assume true for all P_{sv} of k or fewer edges.

Induction step: Let P_{sv} be an $s - v$ path of $k + 1$ edges.
$P_{sv} = P_{su} + (u, v)$ for some u.

$$length(P_{sv}) = length(P_{sv}) + d(u, v) \geq dist_s(u) + d(u, v) \geq dist_s(v)$$

Otherwise, the Ford step could be applied.

□

Implementation

```
ShortestPath(G, s, t):
   Initialize dist[u] = ∞ for all u
   dist[s] = 0
   # queue tracks nodes that are candidates for Ford rule
   queue = [s]
   while queue is not empty:
      v = front of queue (and remove v)
      for w ∈ neighbors(v):
         # Apply Ford rule if we can
         if dist[v] + d(v,w) < dist[w]:
            dist[w] = dist[v] + d(v,w)
            parent[w] = v
            if w ∉ queue: put w at end of queue
```

Running time

- $n =$ number of nodes
- $m =$ number of edges

After $dist_s(v)$ has been updated k times, it corresponds to a path of k edges.

A shortest path can contain at most $n - 1$ edges, so each $dist_s(v)$ can be updated at most $n - 1$ times.

Updating all vertices once takes time $O(m)$ since we look at each edge twice.

Total running time $= O(mn)$.

Note that this is slower than Dikstra's algorithm in general.

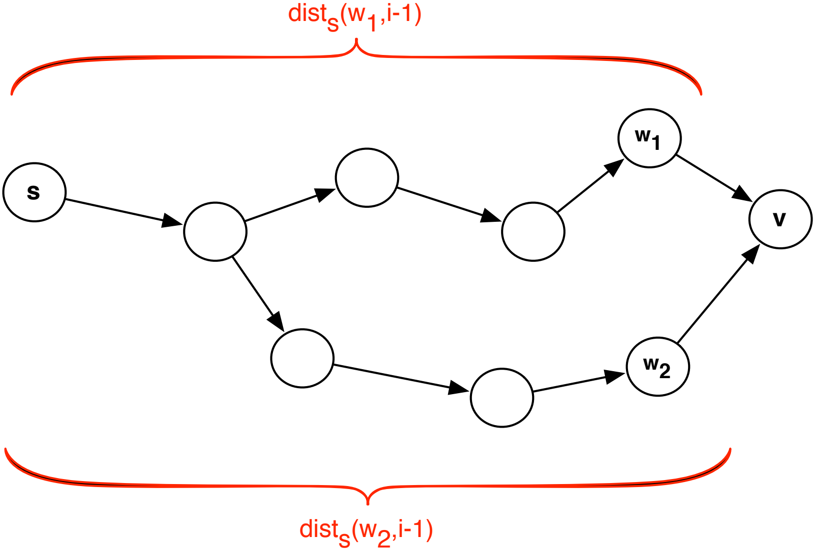
Another view

Definition. Let $dist_s(v, i)$ be minimum cost of a path from s to v that uses at most i edges.

1. If best $s - v$ path uses at most $i - 1$ edges, then $dist_s(v, i) = dist_s(v, i - 1)$.

2. If best $s - v$ uses i edges, and the last edge is (w, v), then $dist_s(v, i) = d(w, v) + dist_s(w, i - 1)$.

Subproblems, picture



Recurrence

Let $N(w)$ be the neighbors of w.

$dist_s(v, i)$ = cost of best path from s to v using at most i edges.

Recurrence:

$$dist_s(v, i) = \min \begin{cases} dist_s(v, i-1) \\ \min_{w \in N(v)} dist_s(w, i-1) + d(w, v) \end{cases}$$

Goal: Compute $dist_s(t, n-1)$.

Code

```
ShortestPath(G=(V,E), s, t):
 Initialize dist_s[x, i] for all x
 For i = 1,...,|V|-1:
   For v in V:
     // find the best w on which to apply the Ford rule
     best_w = None
     for w in N(v):  // N(v) are neighbors of v
        best_w = min(best_w, dist_s[w, i-1] + d[w,v])

     dist_s[v,i] = min(best_w, dist_s[v, i-1])
   EndFor
 EndFor
 Return M[t, n-1]
```

Running Time

Simple Analysis:

- $O(n^2)$ subproblems
- $O(n)$ time to compute each entry in the table (have to search over all possible neighbors w).
- Therefore, runs in $O(n^3)$ time.

A better analysis:

- Let n_v be the number of edges entering v.
- Filling in each entry actually only takes $O(n_v)$ time.
- Total time $= O\left(n \sum_{v \in V} n_v\right) = O(nm)$.